VU Research Portal

Simple product form insights for processor allocation in FMS

van Dijk, N.M.

1991

Faculteit der Economische Wetenschappen en Econometrie

Serie Research Memoranda

Simple Product Form Insights for Processor Allocation in FMS

Nico M. van Dijk

Research Memorandum 1991-83
December 1991

vrije Universiteit *amsterdam*

SIMPLE PRODUCT FORM INSIGHTS
for
PROCESSOR ALLOCATION in FMS

Nico M. van Dijk
Free University, The Netherlands

Abstract

Flexible manufacturing cells are studied with parallel processors, multiple job-types, processor preference lists and job-priorities. The primary purpose of this note is to illustrate that by simple non-mathematical insights one can conclude whether or not the cell exhibits a product form expression for the steady state processor occupancy distribution. Product form results are concluded for various allocation protocols under a

"fall back mechanism".

Additionally, insensitivity results are argued under

"guaranteed random server allocation".

The results illustrate the relevance of

. simple "flow out = flow in" insights and
. an appropriate description of admissible states.

These observations seem of interest in themselves and promising for further practical exploitation. The potential of product form results also for non-product form systems is illustrated by two applications.

Key words

Flexible manufacturing cell * Preference lists * Job priorities * Product forms * Insensitivity * Fail back mechanism * Random allocation.

1 INTRODUCTION

Motivation

Product form results for queueing networks are widely reported in the literature. However, their applicability for specific applications, most notably when job interferences, contention for common resources or parallel processing situations are involved, still appears to be unclear to practitioners. As a representative example of typical practical interest, it doesn't seem to be generally known when and when not multiple job-type allocation procedures such as in flexible manufacturing cells or parallel processor mechanisms exhibit a product form and how this can be concluded with simple insights.

Though such results can in principle be extracted from general abstract frameworks in the literature, most notably [2], [4], [5] and [6], given the non-trivial transformations that would then be required without obtaining simple insights or proofs and given the interests of job- allocation and related structures in practice, a selfcontained investigation at down-to-earth level which would provide simple practical insights seems to be worthwhile in its own right.

Objective

Motivated by the job-allocation problem in FMS-cells, this note merely aims to illustrate the following aspects as of possible interest to practitioners in FMS and parallel processing:

(i) The failure of product form results can frequently be concluded directly by showing that simple physical "flow out = flow in" conditions are violated.

(ii) These simple balance insights also enable one to conclude explicit product form results provided special protocols are in order. Most notably, job-allocation protocols in FMS-cells are shown to exhibit a product-form provided priority lists for both job-allocation and servicing are taken into account.

(iii) Simple insights can also be provided to conclude under which protocols these product forms are insensitive, that is whether these forms remain valid for non-exponential service requirements.

(iv) The product form results can be unrealistic but the insights may still be useful to obtain simple results for non-product form systems.

To highlight the insights rather than the formal mathematics the presentation will be kept rather verbal with instructive examples.

Summary

The results can roughly be summarized by:

(i) Simple notions of partial balance or "flow out = flow in" principles are given by which at system or protocol level one can conclude whether or not a system can have a product form.

(ii) Necessary system rules for product form expressions are so concluded:

. a fall back mechanism . job-dependent rather than server-dependent rates . a server-preference and job-priority listing that uniquely determines the server occupancy

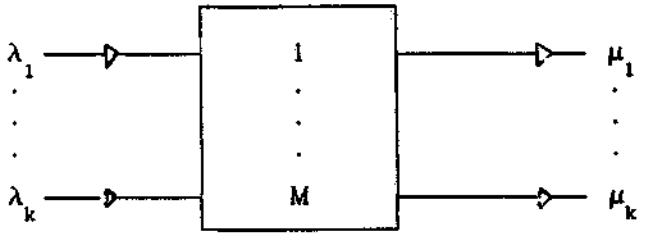
and for additional insensitivity:

. random server allocation . a guaranteed server

(iii) Explicit product form results are obtained, specifying the servers and job-types configuration, which seem to be unreported in the literature as server-preference lists and job-priorities are involved.

Further, two illustrations are given of how the product form insights can also be exploited for non-product form situations. An evaluation concludes the paper.

Model



Consider a service facility with M processors with $M<\infty$ or $M=\infty$ and multiple-type arrivals according to independent Poisson processes with arrival rates λ_k for job-type k. The service requirements are exponential with parameter μ_k for job-type k. Each job-type k has a *"server preference list"*:

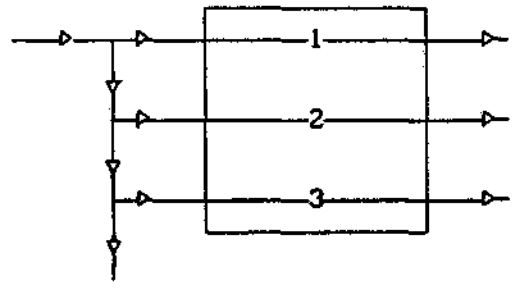
$$P_k = (s_k^1, s_k^2, \ldots)$$

which represents the preference order according to which an arriving type-k job will attempt to find an "*available*" server. (That is, s_k^1 is the first preference, s_k^2 the second, etc.). Here the word "available" may either mean an actual "idle" server (case i below) or a server with a job of lower priority (case ii below) as will be clarified as we proceed. When no "available" server can be found, which takes place instantaneously, the arriving job is lost. (Recall here that we allow $M=\infty$).

2 CASE 1 (DEVOTED PROCESSOR: NO PRODUCT FORM)

Here it is assumed that if a job is assigned a server it has to be completed by that server. More precisely, a service may not be interrupted, nor when meanwhile a higher preference server becomes available, nor when another job arrives. (The latter assumption is merely made for simplicity here). To illustrate what essentially goes wrong in order to conclude a product form, consider the case of only 1 job-type with preference list:

$$S = (1,2,\ldots)$$



Let the state $(s_1,s_2,\ldots)$ denote for all servers i the status s_i of server i, where $s_i=1$ stands for busy and $s_i=0$ for idle, and note that the following sequence of states is possible:

$$(0,0,\ldots) \rightarrow (1,0,\ldots) \rightarrow (1,1,0,\ldots) \rightarrow (0,1,0,\ldots)$$

However, in state (0,1,0,...) the rate out of that state due to server 2 (or the job at server 2) is positive:

$$\text{rate } [(0,1,0,\ldots) \rightarrow (0,0,0,\ldots)] > 0$$

But the rate into state (0,1,0,...) due to server 2 (or the job at server 2) is equal to zero.

$$\text{rate } [(0,0,0,\ldots) \rightarrow (0,1,0,\ldots)] = 0$$

as an arrival in the empty state would have led to: (0,0,0,...)→(1,0,0,...). We thus encounter an

"inconsistency of flow per fixed server (or job)"

This directly tells us that a necessary notion of "partial balance" per server (or per job, or per station if one regards each server as a station) in order to conclude a product form necessarily fails. (E.g. Kelly 79, Schassberger 78, Hordijk and van Dijk 83, Whittle 85). In other words, based on this simple insight of partial flow or rather its failure, no further analysis needs to be employed to conclude that this system cannot have a product form expression.

3 CASE 2: (FLEXIBLE PROCESSORS: PRODUCT FORMS)

The above inconsistency of flow per server was due to the fact that an arriving job did not occupy a lower preference server when a higher preference server was "available". Intuitively, the inconsistency thus seems to be repaired under the following protocol

"When a higher preference server becomes available a job at a lower preference server is to be transformed directly to the higher preference server for its remaining service requirement".

Note here, that more than one job will generally shift to higher preference servers at the same time. Further, for briefness we will refer to this protocol as "*fall back mechanism*".

Under this "fall back mechanism", we will distinguish different situations depending on whether or not also job priorities are involved and if so, whether or not the system is infinite and what priority mechanism is used. In each case, a product form result will be concluded.

3.1 No-job priorities; (model 1)

Here all jobs are assumed to be as equally important to the servers so that an arriving job can never take over a server that is busy. An arriving job is thus assigned the free server of highest preference according to his preference list. When no server of its preference list is found to be free at all (note here that this list can be very restrictive), the job is lost.

Let $\bar{s} = (s_1, s_2, \ldots)$ denote the status s_i of server i, where $s_i=0$ stands for idle, while $s_i=r$ stands for the job at server i to be of type r. Further, let A_1 be the set of admissible states as determined by the preference lists. Then with $\pi(.)$ the steady state distribution and c_1 a normalizing constant at A_1:

(1)
$$\pi(\bar{s}) = c_1 \prod_k \frac{1}{n_k!} \left(\frac{\lambda_k}{\mu_k} \right)^{n_k} \qquad (\bar{s} \in A_1)$$

To prove (1) it suffices to verify the global balance equations. To this end, let $\bar{s}+e_k$ and $\bar{s}-e_k$ denote the state obtained from state $\bar{s}$ by adding or deleting one job of type k, where we note that this state is uniquely determined by virtue of the "fall back mechanism". The global balance equations then become:

(2)
$$\pi(\bar{s}) \left[\sum_k n_k\mu_k + \sum_k \lambda_k 1_{(\bar{s}+e_k \in A_1)} \right] = \sum_k \pi(\bar{s}+e_k)\, \mu_k (n_k+1)\, 1_{(\bar{s}+e_k \in A_1)}] + \sum_k \pi(\bar{s}-e_k)\, \lambda_k$$

where the numbers n_k and n_{k+1} are justified by realizing the "fall-back mechanism" and where $1_{(A)}$ stands for an indicator function of an event A that is $1_{(A)}=1$ if A is satisfied and 1(A)=0 if not. Note that no indicator is needed for states $\bar{s}-e_k$ as $\bar{s}-e_k \in A_1$ for any $\bar{s} \in A_1$ and k such that $s_i=k$ for some i. As these indicator values operate exactly the same in both the left and right hand side of (2), expression (1) is now directly verified by substituting:

(3)

$$\boxed{\begin{aligned} \frac{\pi(\bar{s}+e_k)}{\pi(\bar{s})} &= \left(\frac{\lambda_k}{\mu_k}\right)\frac{1}{n_k+1} \\ \\ \frac{\pi(\bar{s}-e_k)}{\pi(\bar{s})} &= \left(\frac{\mu_k}{\lambda_k}\right) n_k \end{aligned}}$$

Crucially, here it is to be realized, as the preference listings allow jobs to jump from one server to another, that the global balance equations apply as the exponential service rates are *job dependent* and *not server dependent.*

Remark

As in classical queueing models, also a waiting or storage pool may be included for arriving jobs that cannot find a free server. Two possibilities can then be thought of when a server becomes available.

i) In first-come first-order a job from the pool is taken that has this server in his preference list. However, as in the classical FCFS-queueing examples, in order to obtain a simple steady state expression, we then need to assume that all job-types become indistinguishable, that is, with the same preference lists and service parameter μ. In this case it reduces to a standard M/M/c/c+m model with arrival rate $\lambda = \Sigma_k \lambda_k$, with c the total number of servers and with m the size of the storage pool.

ii) Priorities are involved for which jobs from the pool will be brought in service first. Results can then be concluded directly from the next section by simply ignoring the priorities once brought in service. However, as also priorities in the servicing itself then seem to be natural, we will only treat this more complex case in the next section.

3.2 Job-priorities

In this case, job-priorities are allowed by which *higher priority jobs* can *take over servers* from lower priority jobs. Without restriction of generality assume that the job-types are ordered with job-type 1 as highest priority, job-type 2 as second, etc. When a job of type k arrives it searches through its preference list S_k to find a server with highest preference which is "available", meaning that it is either "idle" or that it serves a "lower priority" job. In the latter case this lower priority job instantaneously interrupts its service and searches through his list for an "available" lower preference server in a similar manner. Here the same procedure may have to be followed, etc. etc. A number of jobs may thus shift to lower preference servers all at the same time, provided these can be found. We will distinguish two possibilities below when a job cannot find an "available" server. Conversely, when a server "idles", shifting to higher preference servers will take place of a number of jobs simultaneously according to their preference lists. Let us give an example. Say, we have 3 job-types with preference lists:

$$P_1 = (3,1,4,\ldots)$$
$$P_2 = (1,2,4,\ldots)$$
$$P_3 = (2,4,5,\ldots)$$

Server		Jobs		Jobs		Jobs
s_1	:	2	→	1	→	1
s_2		3		2		2
s_3		1		1		1
s_4		2		2		3
s_5		0		3		0

Essentially, what the priority and preference listings guarantee is that by knowing how many jobs of the various types are present, one knows exactly the system configuration. (That is, which servers are serving which type jobs).

For jobs from which the server is taken over by higher priority which cannot find an "available" server, two possibilities will be distinguished:

a) The job directly clears the system.

b) The job is brought into a common pool

Jobs in this pool will wait until a server idles upon which in order of priority one of these jobs is searched to occupy this server to continue its residual service requirement. If more than one job of this specific priority are present one of these is chosen randomly. Here without restriction of generality we assume that the pool is unrestricted. However, an arriving job is allowed to be routed to the pool directly only if no available server can be found and if at least one job of the same priority class is in service. In other words,

a job is allowed to enter if its
priority class can be worked upon directly

since otherwise

the rate into a state due to that class > 0,
while the rate out of that state due to that class = 0

so that a partial balance notion per job class, as will be required below in order to prove a product form, will be violated. Conversely, if one would reject jobs directly when no available server can be found but with at least one job of that class already in service and one already in the pool,

the rate out of that state due to that class would be > 0
while the rate into that state due to that class would be = 0

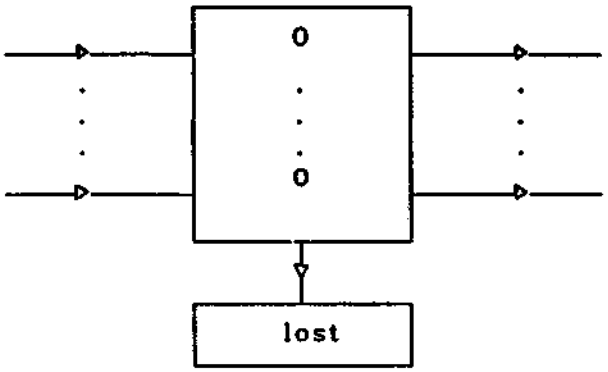
For example, consider two priority classes, 10 servers, and the state with 8 jobs of class 1 and 3 of class 2, of which 2 in service and 1 waiting. Then, the rate out and into that state due to class-2 jobs would have the structure:

$$\begin{pmatrix} 8 \\ 2 \end{pmatrix} \not\longrightarrow \begin{pmatrix} 8 \\ 3 \end{pmatrix} \longrightarrow \begin{pmatrix} 8 \\ 2 \end{pmatrix}$$

Inrate = 0 Outrate > 0

so that also partial balance per job class, in this class-2, would be violated. Partial balance per job-class, as aimed at to conclude a product form, and pooling thus enforce the access protocol given.

a) (Model 2)



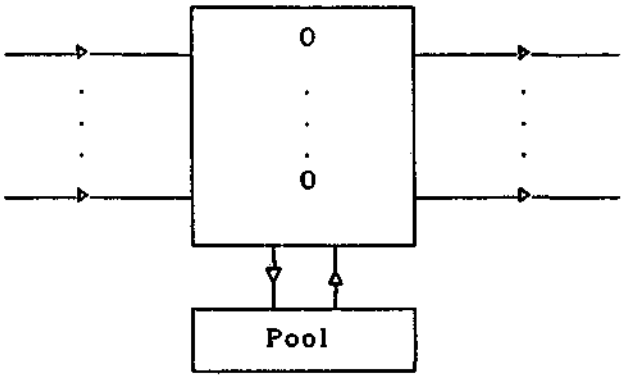
Let $\bar{s} = (s_1, s_2, \ldots)$ be defined as before and let A_2 be the corresponding set of admissible states $\bar{s}$ with not only the preference lists but also the priorities taken into acccount. Now note that by virtue of the priority and preference mechanism also the states $\bar{s}+e_k$ and $\bar{s}-e_k$, representing the state obtained from $\bar{s}$ by adding and deleting one type-k job, are still well-defined in a unique manner. As a consequence, the global balance equations are also given by (2) with A_1 replaced by A_2 so that now with normalizing constant c_2 at A_2:

(4)
$$\pi(\bar{s}) = c_2 \prod_k \frac{1}{n_k!} \left(\frac{\lambda_k}{\mu_k}\right)^{n_k} \qquad (\bar{s} \in A_2)$$

Remark

Note that lower priority jobs might thus receive service but still be lost and not be completed. Roughly speaking, the probability of getting completed will thus be larger for a higher and be lower for a lower priority job as opposed to the case where all jobs would be of same priority.

b) (Model 3)



Now let $(\bar{s},\bar{m})$ denote the description $\bar{s}$ of all servers and the configurations $\bar{m}=(m_1,m_2,\ldots)$ of the numbers m_i of type-i jobs that are contained in the pool. As before, let the admissible configurations of $\bar{s}$ be restricted to some A_3. In addition, the possible pool configurations $\bar{m}$ are restricted to a set C such that

(5) $$\boxed{\bar{m} \in C \Rightarrow \bar{m} - e_k \in C \quad \text{for all } k.}$$

Now note that states $(\bar{s},\bar{m})+e_k$ and $(\bar{s},\bar{m})-e_k$ are still uniquely obtained from $(\bar{s},\bar{m})$ by adding or deleting a type-k job.

As a consequence, again the global balance equations (2) essentially still apply. More precisely, realizing that n_k only represents the number of type-k jobs in the pool and with $m_k[(\bar{s},\bar{m})+e_k]$ the number of type-k jobs in the pool in state $(\bar{s},\bar{m})+e_k$ the global balance equations become:

(6)

$$\boxed{\begin{aligned}&\pi(\bar{s},\bar{m})\left[\Sigma_k\, n_k\mu_k + \Sigma_k\, \lambda_k\, 1_{(n_k>0 \text{ and/or } m_k((\bar{s},\bar{m})+e_k)=0)}\, 1_{((\bar{s},\bar{m})+e_k\in(A_3,C))}\right] = \\ &\Sigma_k\, \pi((\bar{s},\bar{m})+e_k)\, \mu_k\left[n_k+1_{(m_k((\bar{s},\bar{m})+e_k=0)}\right] 1_{((\bar{s},\bar{m})+e_k\in(A_3,C))} + \\ &\Sigma_k\, \pi((\bar{s},\bar{m})-e_k)\lambda_k\, 1_{(n_k>0)}\end{aligned}}$$

In this case, in stead of (3), by substituting

$$\frac{\pi[(\bar{s},\bar{m})+e_k]}{\pi[(\bar{s},\bar{m})]} = \begin{cases} \left(\frac{\lambda_k}{\mu_k}\right)\frac{1}{n_k+1} & \text{if } m_k([\bar{s},\bar{m}]+e_k) = 0 \\ \left(\frac{\lambda_k}{\mu_k}\right)\frac{1}{n_k} & \text{if } m_k([\bar{s},\bar{m}]+e_k) > 0 \\ & \text{and } n_k > 0 \end{cases}$$

(7)

$$\frac{\pi[(\bar{s},\bar{m})-e_k]}{\pi[(\bar{s},\bar{m})]} = \left(\frac{\mu_k}{\lambda_k}\right) n_k \qquad \text{if } m_k([\bar{s},\bar{m}]) = 0,\ n_k > 0$$

and with c_3 in this case the normalizing constant for $[\bar{s},\bar{m}]\in(A_3,C)$ the equations (6) are verified by:

(8)

$$\boxed{\pi(\bar{s},\bar{m}) = c_3 \prod_k \left[\left(\frac{\lambda_k}{\mu_k}\right)^{[n_k+m_k]} \frac{1}{n_k!}\left(\frac{1}{n_k}\right)^{m_k}\right]}$$

Remarks

1. Note that $\bar{n} = (n_1,n_2,\ldots)$, which denotes the numbers n_i of type-i jobs that are in service, is uniquely determined by $\bar{s}$ and conversely. As a consequence, with $\bar{A}_j$ the corresponding sets of admissible states of the form $\bar{n}$ for models 1 and 2 and $(\bar{n},\bar{m})$ for model 3:

(9)

$$\boxed{\begin{aligned} \pi(\bar{n}) &= c_j \prod_k \frac{1}{k!}\left(\frac{\lambda_k}{\mu_k}\right)^{n_k} && (\bar{n}\in\bar{A}_j,\ (j=1,2)) \\ \pi(\bar{n},\bar{m}) &= c_j \prod_k \frac{1}{k!}\left(\frac{\lambda_k}{\mu_k}\right)^{n_k}\left(\frac{1}{n_k}\right)^{m_k} && (\bar{n}\in\bar{A}_3) \end{aligned}}$$

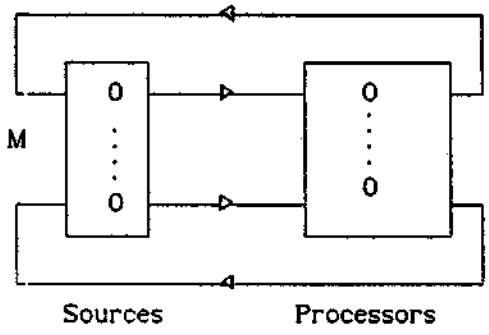
2. The expressions (9) would have been slightly more natural to be proven directly. However, that the same expressions also apply to the more detailed processor distribution is more special. In fact, none of the forms (1), (4), (8) or (9) seems to be reported in the literature.

3. In fact, the equations (2) are verified by partial balance per job-class.

3.4 Some extensions

1) Finite source input (closed case)

As manufacturing systems are frequently modeled as closed queueing systems, first let us also briefly discuss a closed finite source analogue. Here the cell is fed by a finite number of M sources, say with exponential scheduling time for source k with parameter λ_k. A source can only schedule a job to be generated when it has no job in service.



In all situations, the equations (2) remain valid, where however one must realize that each source or job represents one job-type so that the sets A_j restrict to states $\bar{s}$ with at most 1 job of each type. The forms (1), (4) and (8) thus reduce to:

(10)
$$\pi(\bar{s}) = c_j \prod_{\text{(k: source k busy)}} \left[\frac{\lambda_k}{\mu_k}\right]$$

This form may at first glance seem inconsistent with the earlier forms. However, by assuming that for T different classes M_t sources have the same parameters λ_k and μ_k, say λ_t and μ_t, t=1,...,T, and denoting by $\bar{n} = (n_1,n_2,...,n_T)$ the numbers n_t of sources of class t with a job in service, we immediately conclude from (10):

(11)
$$\pi(\bar{n}) = c_j \prod_t \binom{M_t}{n_t} \left(\frac{\lambda_t}{\mu_t}\right)^{n_t}$$

2) Networks of cells

Extensions are possible to networks of the above cells to the extent that the network itself would be of product form if each cell would be of standard form (e.g. like a M|M|1-queue). As, however, the purpose of this paper is to primarily investigate the processor allocation problem, these extensions are not worked out further as they would essentially only adopt results from literature and require much more notational technicalities without providing further results or insight in the allocation problem per cell.

4. NON-EXPONENTIAL CASE: INSENSITIVITY

Insensitivity results are well-known in queueing provided specific notions of partial balance are satisfied (cf. [3], [4], [5], [6]). Here "insensitivity" stands for the property that the same steady state (product form) distribution is retained without the exponential service assumptions provided the same mean service requirements $(1/\mu_k)$ are preserved. As exponentiality conditions are rarely met or verifiable in practice, this property is of significant practical interest. Most notably, in manufacturing applications one should rather think of deterministic service times.

In the present setting the special partial balance required to conclude the insensitivity property comes down to the notion of balance per any fixed job (e.g. [3]) which is to be read as:

(12) *The rate out of a state due to any specific job is equal to the rate into that same state due to that same job.*

However, when priority jobs are involved jobs may shift from server when a job completes service or a new job arrives. To illustrate the consequence of this shifting for keeping track of the same job, as needed for the notion of *balance per job*, consider the following example.

Example Let

$$P_1 = P_2 = (1,2,\ldots)$$

where we assume (as implicitly assumed and used so far):

(13)

When a job comes in he can take over a server from a lower priority job but not from a job of its own priority.

The global balance equations in state (1,2) then become

(14)

$$\pi(1,2)\,[\mu_1 + \mu_2 + \lambda_1 + \lambda_2] =$$
$$\pi(2,0)\lambda_1 + \pi(1,0)\lambda_2 + \pi(1,1,2)2\mu_1 + \pi(1,2,2)2\mu_2$$

which is satisfied by expression (4) in the following detailed manner

(15a) $\pi(1,2)\mu_1 = \pi(2,0)\lambda_1$ (balance of the job at server 1 in (1,2))

(15b) $\pi(1,2)\mu_2 = \pi(2,0)\lambda_2$ (balance of the job at server 2 in (1,2))

(15c) $\pi(1,2)\lambda_1 = \pi(1,1,2)2\mu_1$ (balance of one job 1 against 2 jobs 1)

(15d) $\pi(1,2)\lambda_2 = \pi(1,2,2)2\mu_2$ (balance of one job 2 against 2 jobs 2)

Here the first two relations correspond exactly to the notion of balance per fixed job. For the relations (3) and (4), however, this notion necessarily fails as the rate of one job is balanced by the rate of two jobs. More precisely, according to the above assumed processor allocation rule and the priority of type-1 job over type-2 jobs, the arriving type-1 job in (15c) is assigned server 2, so that the inrate due to the type-1 job at server 1 in state (1,1,2) is equal to: 0. However, as this type-1 job at server 1 in state (1,1,2) is served at rate μ_1 we thus have:

The rate out of state (1,1,2) due to the job at server 1: > 0
The rate into this state (1,1,2) due to this job at server 1: = 0

As a consequence, and based on literature (e.g. [3], [5]):

(16)

Under (13)
the notion of balance per job
and thus the insensitivity property fail.

However, by assuming that the arriving job in (15c) is assigned

(17)
$$\begin{cases} \text{server 1 with probability } 1/2 \\ \text{server 2 with probability } 1/2 \end{cases}$$

where in the first case the old type-1 job at server 1 is shifted to server 2, (15c) can be decomposed and reinterpreted as:

(18)
$$\begin{aligned} &\pi(1,2)(1/2)\lambda_1 = \pi(1,1,2)\,\mu_1 \quad \text{(balance of the job at server 1 in state (1,1,2))} \\ &\pi(1,2)(1/2)\lambda_1 = \pi(1,1,2)\,\mu_1 \quad \text{(balance of the job at server 2 in state (1,1,2))} \end{aligned}$$

so that also (15c) actually comes down to balance per job. This insight will lead to the processor allocation rules below in order to conclude insensitivity results.

4.1 Model 2: Priorities, no pool.

Reconsider model 2 with priorities and no pool. But in stead of (13), as assumed before, we now assume the following generalization of (17):

Random processor allocation per type

(19)

When a type-k job arrives in a state $\bar{s}$ with n_k type-k jobs present and the state $\bar{s}+e_k$ is admissible, say with servers $p_1^k, p_2^k, \ldots, p_{n_k+1}^k$ for type-k jobs, in order of the k-th preference list one of these servers is randomly assigned to this new job, that is server p_i with probability $1/[n_k+1]$, $i=1,\ldots,n_{k+1}$. In that case the type-k jobs formerly at servers $p_1^k,\ldots,p_{n_k}^k$ are shifted to servers $p_{i+1}^k,\ldots,p_{n_k+1}^k$.

The global balance equations (2) can be written as:

(20)

$$\pi(\bar{s}) \sum_p \mu_{s_p} + \sum_k \lambda_k 1_{(\bar{s}+e_k \in A_2)} =$$

$$\sum_k \pi(\bar{s}+e_k) 1_{(\bar{s}+e_k \in A_2)} [n+1] \mu +$$

$$\sum_p \pi(\bar{s}-e_k) \lambda_k [1/n_k]$$

These are satisfied by expression (4) in the following detailed manner of balance per job:

(21)

$$\pi(\bar{s}) \mu_{s_p} = \pi(\bar{s}-e_k) \lambda_k 1/[n_k]$$ (balance for the type-k job at server p, where $s_p=k$, $p=p_1^k,\ldots,p_{n_k}^k$) ($k=1,2,\ldots.$)

$$\pi(\bar{s}) \lambda_k 1/[n_k+1] = \pi(\bar{s}+e_k) \mu_k$$ (balance for the arriving type-k job, assigned the specific server p_i^k) ($i=1,\ldots,n_k+1$)

Consequently:

(22)

> In any state and for any job: balance per that job as per (12) is valid.

From [3], [5] or [6] we can then conclude:

(23)

> Under the random processor allocation per type as per (19) the product-form expression (4) is insensitive, that is holds for arbitrary service requirements with means $1/\mu_k$ for type k, where after shifting to another server the service of a job is continued to receive the residual amount of service required.

4.2 Model 3 : Priorities and pool

Here reconsider model 3 with priorities and a pool but with (13) modified as will be specified below. To this end, however, let us first realize that the notion of balance per job (12) necessarily requires the condition of "instantaneous attention" (also see [5]):

(24)

An arriving job which is accepted needs to receive a positive service rate instantaneously

since otherwise the rate into a state due to an arriving job would be positive while the rate out of that state due to that job would be 0. This directly leads to the conclusion:

(25)

When an arriving job can be assigned to the pool directly (with positive probability that is), the system cannot be insensitive.

Therefore, instead of (13), as assumed before, and also taking into account (16) as argued before, we now assume the following processor allocation rule:

Random allocation per type and not to the pool

(26)

When a type-k job arrives in a state $(\bar{s},\bar{m})$ with n_k type-k jobs in service, say with servers $p_1,p_2,\ldots,p_{n_k}$ in order of the k-th preference list, we can have the following two possibilities:

i) The state $\bar{s}+e_k$ is *admissible* with extra type-k server $p_{n_{k+1}}$
In this case, the job is assigned any of these servers p_i with probability $1/[n_k+1]$, $i=1,\ldots,n_k+1$. When assigned p_i the type-k jobs previous at $p_1,\ldots,p_{n_k}$ are shifted to $p_{i+1},\ldots,p_{n_k+1}$.

ii) The state $s+e_k$ is *not admissible*. In this case, the job is assigned any of the servers p_i with probability $1/[n_k]$, $i=1,\ldots,n_k$. When assigned server p_i the type-k jobs previously at $p_1,\ldots,p_{n_k-1}$ are shifted to $p_{i+1},\ldots,p_{n_k}$ while the job at server p_{n_k} is shifted to the pool. When $\bar{m}+e_k \notin C$ one of the type-k jobs in the pool is randomly chosen, possibly the latter job, to clear the system.

The global balance equations (6) can now be rewritten as:

$$\pi(\bar{s},\bar{m})\left[\sum_p \mu_{s_p} + \sum_k \lambda_k 1_{([\bar{s},\bar{m}]+e_k \in (A_3,C))}\right] =$$

(27) $$\sum_k \pi([(\bar{s},\bar{m})+e_k])\mu_k\,[n_k+1_{(m_k([\bar{s},\bar{m}]+e_k)=0)}]1_{([\bar{s},\bar{m}]+e_k \in (A_3,C))} +$$

$$\sum_p \pi([\bar{s},\bar{m}]-e_k)\,\lambda_k[1/n_k]$$

Similarly to (21), these in turn are satisfied by expression (8), by using (7), in the following manner of balance per job:

$$\pi(\bar{s},\bar{m})\,\mu_{s_p} = \pi([\bar{s},\bar{m}]-e_k)\,\lambda_k[1/n_k]$$ (balance for the type k-job at server p where $s_p=k$) $(p=p_1^k,\ldots,p_{n_k}^k)$

(28) $$\pi(\bar{s},\bar{m})\,\lambda_k[1/[n_k+1] = \pi(\bar{s}+e_k,\bar{m})\,\mu_k$$ (balance for the arriving type-k job, assigned the specific server p_i^k) $(i=1,\ldots,n_k+1)$, if $\bar{s}+e_k \in A_3$)

$$\pi(\bar{s},\bar{m})\,\lambda_k[1/n_k] = \pi([\bar{s},\bar{m}+e_k])\,\mu_k$$ (as above but with $i=1,\ldots,n_k$ when $\bar{s}+e_k \notin A_3$ but $\bar{m}+e_k \in C$).

Note that all rates of (27) are hereby covered. Or more explicitly, note that both the rate out of and into any state $[\bar{s},\bar{m}]$ due to any job in the pool is equal to 0 by virtue of the server allocation protocol that always instantaneously assigns an arriving job to a server. Again, we have thus concluded.

(29) In any state and for any job: balance per job as per 12 is valid.

As before, (see (23)), we can now conclude from literature ([3], [5] or [6]):

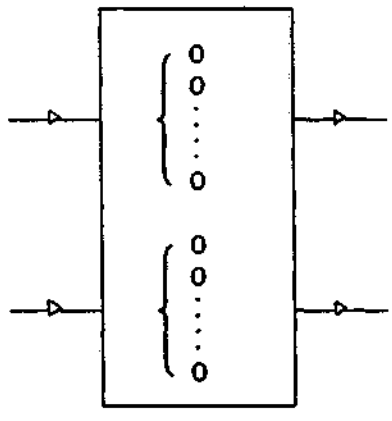
(30) Under the random allocation per type which always assigns a server to an accepted job as per (26), the product form expression (8) is insensitive.

5 TWO NON-PRODUCT FORM APPLICATIONS

Clearly, the allocation rules to meet the partial balance conditions may not be satisfied in specific practical applications. Most notably, the "fall back mechanism" will typically be violated whenever job-services cannot be interrupted. This section aims to illustrate that the product form results andthe insights obtained might also be useful in such situations.

The system under investigation belongs to the category of model 2. More precisely, we consider a parallel processor system with two job-types, N_1 processors of class 1 and N_2 processors of class 2. Both job-types have highest service preference for class 1. Type-1 jobs can only be handled by type-1 servers and have priority over type-2 jobs. It is assumed that a job cannot be shifted back from a class 2 to a class 1 server. That is, the "fall back mechanism" does not apply. When no server is available an arriving job is lost, that is:

$$\text{for job-type 1 when } n_1 = N_1 \text{ and}$$
$$\text{for job-type 2 when } n_2 = N_1 + N_2 - n_1.$$



5.1 Exact expression

Let $\bar{\lambda}_i$ be the real throughput of type-i jobs, that is the mean number of type-i jobs that are serviced by the system per unit of time. One directly notes that $\bar{\lambda}_1$ is equal to the throughput of the classical $M/M/N_1$ loss system with parameters λ_1, μ_1 since type-2 jobs do not interfere with type-1 jobs. But no equivalent statement applies for $\bar{\lambda}_2$.

However, as the servicing of type-2 jobs is determined purely by their number present and not the servers that actually serve them, the throughput $\bar{\lambda}_2$ of this system has to be equal to that under the "fall back mechanism". In other words, $\bar{\lambda}_2$ can be computed by using the product form expression (4) as by:

(31) $$\bar{\lambda}_2 = \sum_{\bar{s}} \pi(\bar{s})\, n_2 \mu_2$$

5.2 Simple bounds

As another performance measure of interest one might wish to investigate the mean number of busy class-2 servers, say denoted by B. Now let us recall, as illustrated by the example in section 2, that the necessary partial balance per server (or job) failed as:

> the rate out of a busy class-2 server is always positive
> while the rate into this server is zero
> when a class-1 server is free.

One way to "repair" or rather avoid this inconsistency is the "fall back mechanism", which leads to the expression (4). A second way to repair it is by:

> stop service of class-2 servers when a class-1 server is free

Indeed, with n_2^1 the number of type-2 jobs at class-1 servers and n_2^2 the number of type-2 jobs at class-2 servers, under this protocol one easily verifies by substitution in the global balance equations:

(32)
$$\pi(n_1, n_2^1, n_2^2)$$
$$\left[n_1\mu_1 + n_2^1\mu_2 + n_2^2\,\mu_2\, 1_{\{n_1+n_2^1 = N_1\}} + \lambda_1 1_{\{n_1<N_1\}} + \right.$$
$$\left. \lambda_2\, 1_{\{n_1+n_2^1<N_1\}} + \lambda_2\, 1_{\{n_1+n_2^1 = N_1\}}\, 1_{\{n_1+n_2^1+n_2^2<N_1+N_2\}} \right]$$
$$=$$
$$\pi(n_1-1, n_2^1, n_2^2)\,\lambda_1 + \pi(n_1, n_2^1-1, n_2^2)\,\lambda_2 + \pi(n_1, n_2^1-1)\,\lambda_2\, 1_{\{n_1+n_2^1 = N_1\}} +$$
$$\pi(n_1+1, n_2^1, n_2^2)(n_1+1)\, 1_{\{n_1<N_1\}} + \pi(n_1, n_2^1+1, n_2^2)\,(n_2^1+1)\,\mu_2\, 1_{\{n_1+n_2^1<N_1\}} +$$
$$\pi(n_1, n_2^1, n_2^2+1)(n_2^2+1)\,\mu_2\, 1_{\{n_1+n_2^1 = N_1\}}\, 1_{\{n_1+n_2^1+n_2^2 < N_1+N_2\}}$$

the product form expression

$$\pi(n_1,n_2^1,n_2^2) = c\,\frac{1}{n_1!}\left(\frac{\lambda_1}{\mu_1}\right)^{n_1}\frac{1}{n_2^1!}\left(\frac{\lambda_1}{\mu_1}\right)^{n_2^1}\frac{1}{n_2^2!}\left(\frac{\lambda_2}{\mu_2}\right)^{n_2^2} \tag{33}$$

On the other hand, the fall mechanism clearly gives a lower bound B_L for B while the latter modification gives an upper B_U. Hence, with

B_U calculated by (33)
B_L calculated by (4)

we have:

$$B_L \le B \le B_U \tag{34}$$

Some numerical illustration indicates that these simple bounds give robust but quick and secure estimates of the order of magnitude and reasonable estimates by using their middle value $(B_L+B_U)/2$.

Acknowledgement

The author is most grateful to Paul Schweitzer for stimulating discussions on the processor allocation problem which motivated this paper.

References

[1] Buzacott J.A. and Yao D.D. (1986), "On queueing network models of flexible manufacturing systems", *Queueing Systems* 1, 29-66.

[2] Hordijk, A. and Van Dijk, N.M. (1983), "Networks of queues. Part I: Job-local-balance and the adjoint process. Part II: General routing and service characteristics", Lecture notes in control and Informational Sciences, *Springer-Verlag*, 60, 158-205.

[3] Hordijk, A. and Van Dijk, N.M. (1983), "Adjoint processes, job-local-balance and insensitivity of stochastic networks", Bull. 44-th Session *Int. Stat. Inst.*, 50, 776-788.

[4] Kelly, F.P. (1979), "Reversibility and stochastic networks", *Wiley*.

[5] Schassberger, R. (1978), "The insensitivity of stationary probabilities in networks of queues", *Adv. Appl. Prob.* 10, 906-912.

[6] Whittle, P., (1986), "Systems in stochastic equilibrium", *Wiley*.

[7] Yao, D.D. and Buzacott, J.A. (1985), "Modeling a class of state dependent routing in flexible manufacturing systems", *Ann. Oper Res.* 3, 153-167.

[8] Yao, D.D. and Buzacott, J.A. (1985), "Queueing models for a flexible machining station station Part ii: the method of Co phases", *Eur. J. Oper.Res.* 19, 241-252.

[9] Yao, D.D. and Buzacott, J.A. (1987), "Modeling a class of flexible manufacturing systems with reversible routing", *Oper. Res.* 35, 87-93.

Numerical illustration of (34): $B_L \leq B \leq B_U$

B: Mean number of busy class-2 servers ($\mu_1=\mu_2=1$)

N_1	N_2	λ_1	λ_2	B_L	B_U
1	1	1	1	.54	.67
1	1	3	3	.65	.75
1	1	10	5	.79	.83
1	1	5	10	.86	.91
2	2	5	5	1.45	1.62
2	2	10	5	1.52	1.62
2	2	5	10	1.68	1.81
3	3	3	3	1.37	1.96
3	3	5	5	2.02	2.35
3	3	10	5	2.18	2.35
3	3	10	10	2.55	2.68
5	5	5	5	2.66	3.57
5	5	10	5	3.15	3.57
5	5	10	10	4.09	4.36